U-IMPACTIFY

Software Design Documentation

Team Boundless:

- Navinn Ravindaran (`ravindar`)
- Clara Chick (`chickcla`)
- Winson Yuan (`yuanwins`)
- Brian Kim (`kimbri15`)
- Samyak Mehta (`mehtas28`)
- Divyam Patel (`pate1006`)
- Aryan Patel (`pate1065`)

Table of Contents

CRC Cards

Frontend Pages

Class: `LoginSignup`

Responsibilities	Collaborators
- Contains form for the user to submit their login/signup information	- `UserService`
- Gives user the option to login/signup with other methods	

Class: `Questionaire`

Responsibilities	Collaborators
- Display questions based on the type of user you signup as	- `UserService`
- Gather information based on questions submited	

Class: `Dashboard`

Responsibilities	Collaborators
- Display all pages	- `UserService`
- Used to quickly navigate through different pages	- `CourseService`
	- `AuthGuard`

Class: `Course`

Responsibilities	Collaborators
- Display Course information	- `UserService`
- Allow possible interaction with course if `User` has permission	- `CourseService`

Class: `Messaging`

Responsibilities	Collaborators
- Enable Users to send private messages to each other	- `UserService`

Class: `Feedback`

Responsibilities	Collaborators
- Display feedback for courses from registered users	- `UserService`
	- `CourseService`

Class: `GivingGarden`

Responsibilities	Collaborators
- Donate money to a non-profit organization	- `UserService`
- Recieve Funding from individuals and larger organizations	
- Support Impact Learners in courses financially	

Frontend Services

Class: `UserService`

Responsibilities	Collaborators
- Create an account	- `User` (Routes)
- Delete an account	
- Get current user informations	

Class: `CourseService`

Responsibilities	Collaborators
- Create a course	- `Course` (Routes)
- Modify a course	
- Delete a course	
- Upload a file	
- Provide overview of course content	

Frontend Guards

Class: `AuthGuard`

Responsibilities	Collaborators
- Check if User is logged in, if so, allow them to use the site	- `UserService`

Frontend Components

Class: `DashboardCoursesComponent` (only for impact learner and impact consultant)

Responsibilities	**Collaborators**
- Display the courses the user is taking/teaching	- `User`
- Redirects you to create a course for easy access	

Class: `CreateCourseComponent` (only for impact consultant)

Responsibilities	**Collaborators**
- Contains a form for the Instructor to create a new course	- `UserService`
	- `CourseService`

Class: `FrontPageHeaderComponent`

Responsibilities	**Collaborators**
- Display Sign-in and login-in buttons	- `UserService`
- Easy accessible application infromation	

Class: `FooterComponent`

Responsibilities	**Collaborators**
- Display contact information	
- Links to social media accounts	
- Provide links to legal informations	

Class: `GlobalSearchComponent`

Responsibilities	**Collaborators**
- Search for Userss	- `UserService`
- Search for Courses	- `CourseService`
- Search for Documents	

Backend REST API Routes

Route: `/user`

Responsibilities	Collaborators
- Standard CRUD Operations on the User model	- `UserService`
- Return formatted JSON data as called by `UserService`	

Route: `/course`

Responsibilities	Collaborators
- Standard CRUD Operations on the Course model	- `CourseService`
- Upload and Receive documents and/or media files from client to database and v.v	
- Return formatted JSON data as called by `CourseService`	

Meeting the MVC Spec.

> A note on Angular.

Information used in this note was primary source from:
`https://stackoverflow.com/questions/35762515/is-angular2-mvc`

Introduction

For the duration of this project, this team will be using Angular 10 as the front-end framework of choice. Before Angular 10 there was AngularJS, which was, and still is, a MVW (Model, View, Whatever) framework. For our purposes, we can classify it as a pure MVC framework because you can clearly defined View, (`HTML`), Model and Controller (`JavaScript/TypeScript`) files. Angular 10 (or any version after 1.0), is a complete architectural overhaul, with a switch to a more *component-driven architecture*, a CLI, and more.

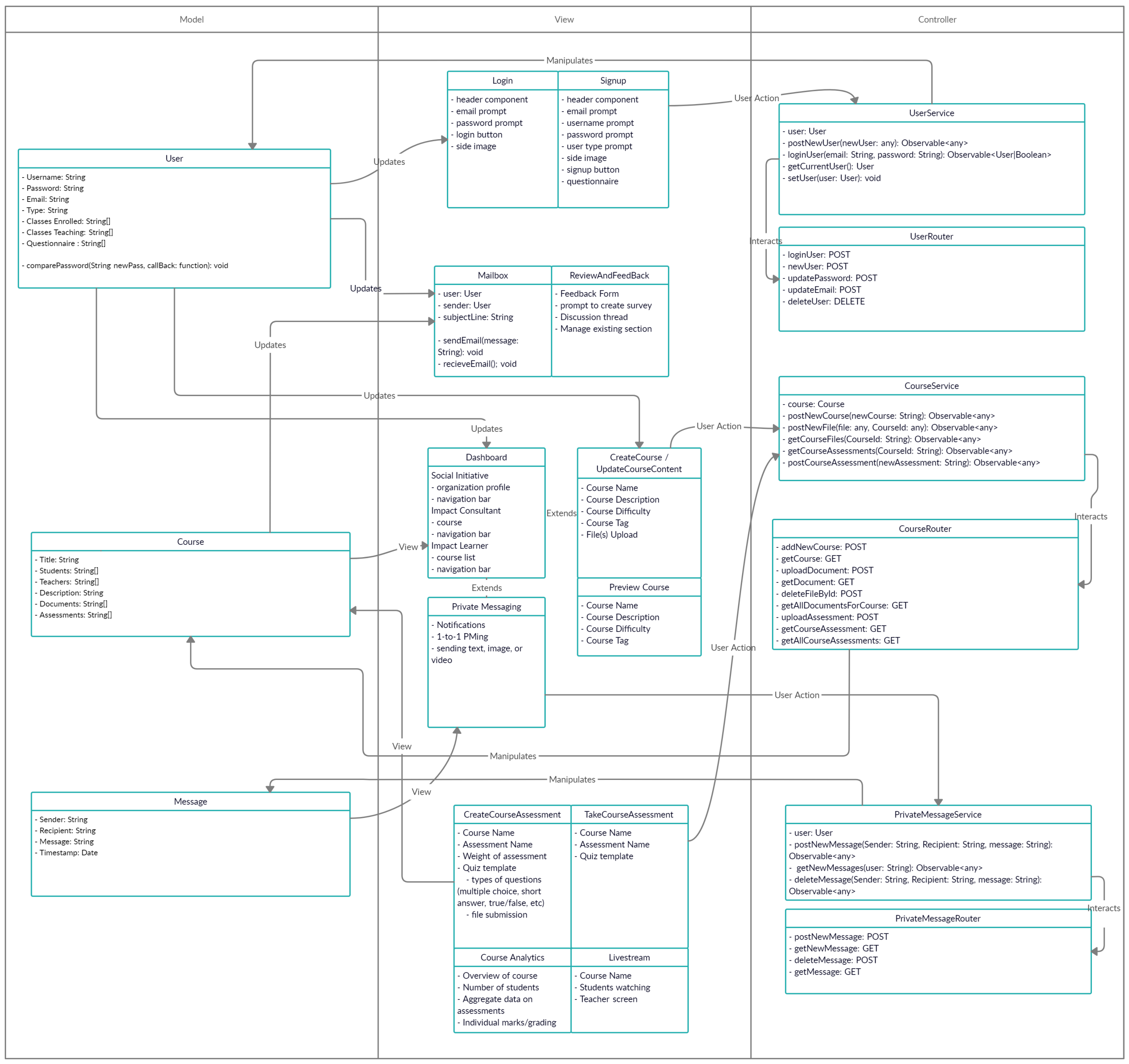
What does this have to do with MVC?

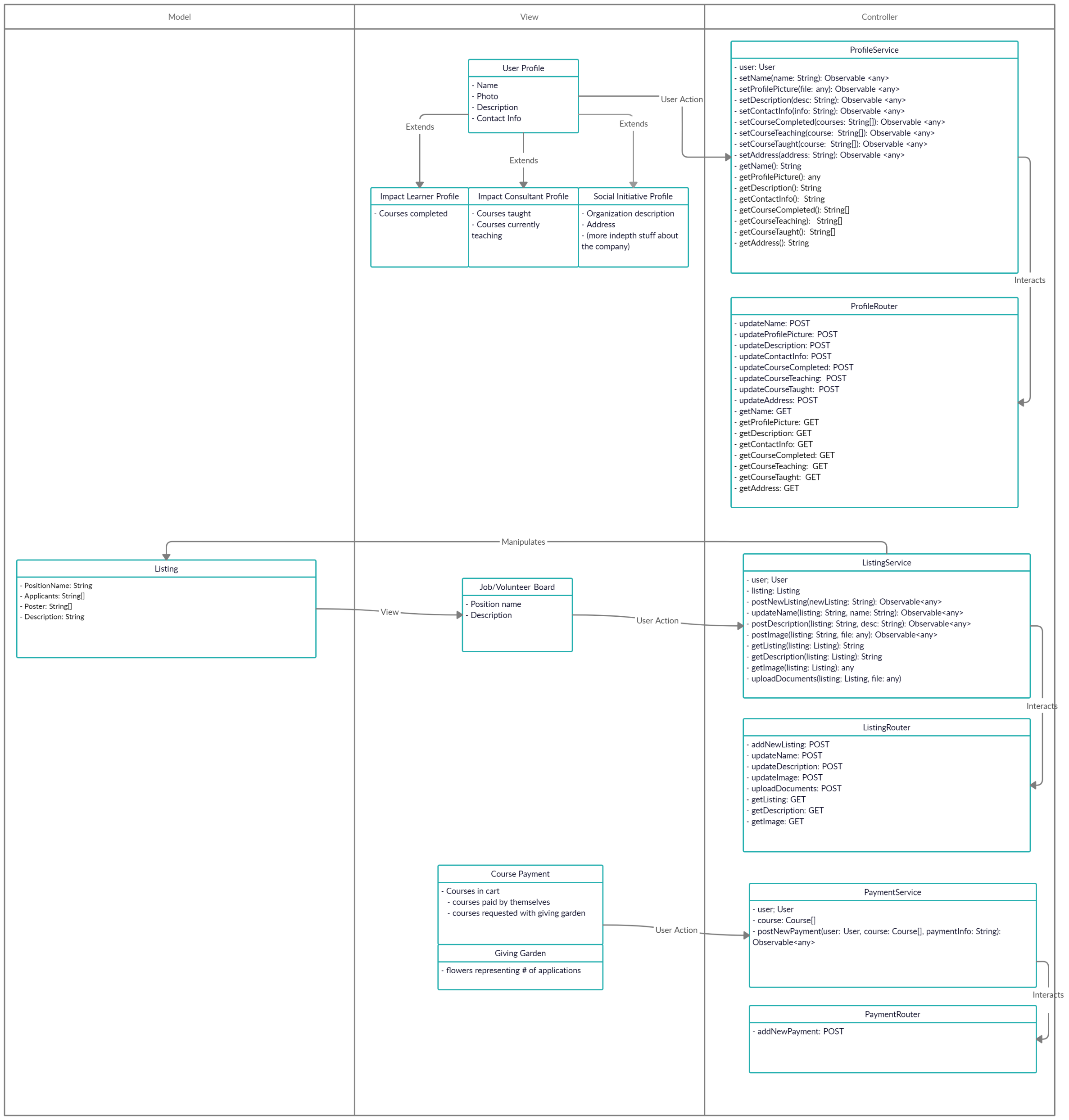
The argument is quite simple, *show Angular 10 follows an MVC based architecture*. Now for the proof, we must consider two different Software Architectures, one for the program as a whole, and one ***specifically*** for each Angular component. For the former, refer to the below images.

In short, we treat models as `mongooseSchemas` (models) in the database files and front-end models that are a copy of the backend model, views as the `html` presented to the user, and controllers as front-end services + backend routes.

Each Angular component gets its own directory with 2 important files (we can disregard the styles and the unit testing files for our purposes), an `html`, and a `ts` file. The `html` file is trivially the view, and with the `ts` file, if its a simple component, we can think of it as ***both*** a controller and model. As input/output logic is handled through functions declared in this file, and model states can also be managed as well, albeit it may not be as organized as it would be using Angular Services. Hence achieving our desired MVC architechture.

Software Architecture Design




Model
View
Controller
User Profile
- Name
- Photo
- Description
- Contact Info
Extends
Extends
Extends
User Action
Impact Learner Profile
- Courses completed
Impact Consultant Profile
- Courses taught
- Courses currently teaching
Social Initiative Profile
- Organization description
- Address
- (more indepth stuff about the company)
ProfileService
- user: User
- setName(name: String): Observable <any>
- setProfilePicture(file: any): Observable <any>
- setDescription(desc: String): Observable <any>
- setContactInfo(info: String): Observable <any>
- setCourseCompleted(courses: String[]): Observable <any>
- setCourseTeaching(course: String[]): Observable <any>
- setCourseTaught(course: String[]): Observable <any>
- setAddress(address: String): Observable <any>
- getName(): String
- getProfilePicture(): any
- getDescription(): String
- getContactInfo(): String
- getCourseCompleted(): String[]
- getCourseTeaching(): String[]
- getCourseTaught(): String[]
- getAddress(): String
Interacts
ProfileRouter
- updateName: POST
- updateProfilePicture: POST
- updateDescription: POST
- updateContactInfo: POST
- updateCourseCompleted: POST
- updateCourseTeaching: POST
- updateCourseTaught: POST
- updateAddress: POST
- getName: GET
- getProfilePicture: GET
- getDescription: GET
- getContactInfo: GET
- getCourseCompleted: GET
- getCourseTeaching: GET
- getCourseTaught: GET
- getAddress: GET
Manipulates
Listing
- PositionName: String
- Applicants: String[]
- Poster: String[]
- Description: String
View
Job/Volunteer Board
- Position name
- Description
User Action
ListingService
- user; User
- listing: Listing
- postNewListing(newListing: String): Observable<any>
- updateName(listing: String, name: String): Observable<any>
- postDescription(listing: String, desc: String): Observable<any>
- postImage(listing: String, file: any): Observable<any>
- getListing(listing: Listing): String
- getDescription(listing: Listing): String
- getImage(listing: Listing): any
- uploadDocuments(listing; Listing, file: any)
Interacts
ListingRouter
- addNewListing: POST
- updateName: POST
- updateDescription: POST
- updateImage: POST
- uploadDocuments: POST
- getListing: GET
- getDescription: GET
- getImage: GET
Course Payment
- Courses in cart
- courses paid by themselves
- courses requested with giving garden
Giving Garden
- flowers representing # of applications
User Action
PaymentService
- user; User
- course: Course[]
- postNewPayment(user: User, course: Course[], paymentInfo: String): Observable<any>
Interacts
PaymentRouter
- addNewPayment: POST